# 工作站 RedHat 教材

## Chapter 1. Improve Command-line Productivity

\* Write Simple Bash Scripts

\* Guided Exercise: Write Simple Bash Scripts

\* Loops and Conditional Constructs in Scripts

\* Guided Exercise: Loops and Conditional Constructs in Scripts

\* Match Text in Command Output with Regular Expressions

\* Guided Exercise: Match Text in Command Output with Regular Expressions

\* Lab: Improve Command-line Productivity

\* Summary

![](https://hackmd.io/\_uploads/SJHsL3Se6.png)

### Write Simple Bash Scripts

#### Objectives

Run commands more efficiently by using advanced features of the Bash shell, shell scripts, and various Red Hat Enterprise Linux utilities.

### Create and Execute Bash Shell Scripts

You can accomplish many system administration tasks by using command-line tools. More complex tasks often require chaining together multiple commands that pass results between them. By using the Bash shell environment and scripting features, you can combine Linux commands into shell scripts to solve repetitive real-world problems.

A Bash shell script is an executable file that contains a list of commands, and possibly with programming logic to control decision-making in the overall task. When well-written, a shell script is a powerful command-line tool on its own, and you can use it with other scripts.

Shell scripting proficiency is essential for system administrators in any operational environment. You can use shell scripts to improve the efficiency and accuracy of routine task completion.

Although you can use any text editor, advanced editors such as vim or emacs understand Bash shell syntax and can provide color-coded highlighting. This highlighting helps to identify common scripting errors such as improper syntax, unmatched quotes, parentheses, brackets, and braces, and other structural mistakes.

### Specify the Command Interpreter

The first line of a script begins with the #! notation, which is commonly referred to as she-bang or hash-bang, from the names of those two characters, sharp or hash and bang. This notation is an interpreter directive to indicate the command interpreter and command options to process the remaining lines in the file. For Bash syntax script files, the first line is the following directive:

#!/usr/bin/bash

### Execute a Bash Shell Script

A shell script file must have execute permissions to run it as an ordinary command. Use the chmod command to modify the file permissions. Use the chown command, if needed, to grant execute permission only for specific users or groups.

If the script is stored in a directory that is listed in the shell's PATH environmental variable, then you can run the shell script by using only its file name, similar to running compiled commands. Because PATH parsing runs the first matching file name that is found, always avoid using existing command names to name your script files. If a script is not in a PATH directory, then run the script by using its absolute path name, which you can determine by querying the file with the which command. Alternatively, run a script in your current working directory by using the . directory prefix, such as ./﻿scriptname.

[user@host ~]$ which hello

~/bin/hello

[user@host ~]$ echo $PATH

/home/user/.local/bin:/home/user/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin

### Quote Special Characters

Some characters and words have a special meaning to the Bash shell. To use these characters for their literal values, rather than for their special meanings, you escape them in the script. Use the backslash character (\), single quotes (''), or double quotes ("") to remove (or escape) the special meaning of these characters.

The backslash character removes the special meaning of the single character that immediately follows the backslash. For example, to use the echo command to display the # not a comment literal string, the # hash character must not be interpreted as a comment.

The following example shows the backslash character (\) modifying the hash character so it is not interpreted as a comment:

[user@host ~]$ echo # not a comment

[user@host ~]$ echo \# not a comment

# not a comment

To escape more than one character in a text string, either use the backslash character multiple times, or enclose the whole string in single quotes ('') to interpret literally. Single quotes preserve the literal meaning of all characters that they enclose. Observe the backslash character and single quotes in these examples:

[user@host ~]$ echo # not a comment #

[user@host ~]$ echo \# not a comment #

# not a comment

[user@host ~]$ echo \# not a comment \#

# not a comment #

[user@host ~]$ echo '# not a comment #'

# not a comment #

Use double quotation marks to suppress globbing (file name pattern matching) and shell expansion, but still allow command and variable substitution. Variable substitution is conceptually the same as command substitution, but might use optional brace syntax. Observe the following examples of various quotation mark forms.

Use single quotation marks to interpret all enclosed text literally. Besides suppressing globbing and shell expansion, single quotation marks also direct the shell to suppress command and variable substitution. The question mark (?) is included inside the quotations, because it is a metacharacter that also needs escaping from expansion.

[user@host ~]$ var=$(hostname -s); echo $var

host

[user@host ~]$ echo "\*\*\*\*\* hostname is ${var} \*\*\*\*\*"

\*\*\*\*\* hostname is host \*\*\*\*\*

[user@host ~]$ echo Your username variable is \$USER.

Your username variable is $USER.

[user@host ~]$ echo "Will variable $var evaluate to $(hostname -s)?"

Will variable host evaluate to host?

[user@host ~]$ echo 'Will variable $var evaluate to $(hostname -s)?'

Will variable $var evaluate to $(hostname -s)?

[user@host ~]$ echo "\"Hello, world\""

"Hello, world"

[user@host ~]$ echo '"Hello, world"'

"Hello, world"

### Provide Output from a Shell Script

The echo command displays arbitrary text by passing the text as an argument to the command. By default, the text is sent to standard output (STDOUT). You can send text elsewhere by using output redirection. In the following simple Bash script, the echo command displays the "Hello, world" message to STDOUT, which defaults to the screen device.

[user@host ~]$ cat ~/bin/hello

#!/usr/bin/bash

echo "Hello, world"

[user@host ~]$ hello

Hello, world

### Note

This user can run hello at the prompt because the ~/bin (/home/user/bin) directory is in the user's PATH variable and the hello script has executable permission. The PATH parser finds the script first, if no other executable file called hello is found in any earlier PATH directory. Your home directory's bin subdirectory is intended to store your personal scripts.

The echo command is widely used in shell scripts to display informational or error messages. Messages helpfully indicate a script's progress, and can be directed to standard output or standard error, or be redirected to a log file for archiving. When you display error messages, good programming practice is to redirect error messages to STDERR to separate them from normal program output.

[user@host ~]$ cat ~/bin/hello

#!/usr/bin/bash

echo "Hello, world"

echo "ERROR: Houston, we have a problem." >&2

[user@host ~]$ hello 2> hello.log

Hello, world

[user@host ~]$ cat hello.log

ERROR: Houston, we have a problem.

The echo command is also helpful to debug a problematic shell script. Adding echo statements in a script, to display variable values and other runtime information, can help to clarify how a script is behaving.

## Guided Exercise: Write Simple Bash Scripts

In this exercise, you write a simple Bash script with a sequence of commands and run it from the command line.

### Outcomes

\* Write and execute a simple Bash script.

\* Redirect the output of a simple Bash script to a file.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

[student@workstation ~]$ lab start console-write

### Instructions

1. Log in to the servera machine as the student user.

[student@workstation ~]$ ssh student@servera

...output omitted...

[student@servera ~]$

2. Create and execute a simple Bash script.

1. Use the vim command to create the firstscript.sh file under your home directory.

[student@servera ~]$ vim firstscript.sh

1. Insert the following text, and save the file. The number of hash signs (#) is arbitrary.

#!/usr/bin/bash

echo "This is my first bash script" > ~/output.txt

echo "" >> ~/output.txt

echo "#####################################################" >> ~/output.txt

1. Use the bash command to execute the script.

[student@servera ~]$ bash firstscript.sh

Review the output file that the script generated.

[student@servera ~]$ cat output.txt

This is my first bash script

#####################################################

1. Add more commands to the firstscript.sh script, execute it, and review the output.

1. Use the Vim text editor to edit the firstscript.sh script.

[student@servera ~]$ vim firstscript.sh

The following output shows the expected content of the firstscript.sh file:

#!/usr/bin/bash

#

echo "This is my first bash script" > ~/output.txt

echo "" >> ~/output.txt

echo "#####################################################" >> ~/output.txt

echo "LIST BLOCK DEVICES" >> ~/output.txt

echo "" >> ~/output.txt

lsblk >> ~/output.txt

echo "" >> ~/output.txt

echo "#####################################################" >> ~/output.txt

echo "FILESYSTEM FREE SPACE STATUS" >> ~/output.txt

echo "" >> ~/output.txt

df -h >> ~/output.txt

echo "#####################################################" >> ~/output.txt

1. Make the firstscript.sh file executable by using the chmod command.

[student@servera ~]$ chmod a+x firstscript.sh

Execute the firstscript.sh script.

[student@servera ~]$ ./firstscript.sh

Review the output file that the script generated.

[student@servera ~]$ cat output.txt

This is my first bash script

#####################################################

LIST BLOCK DEVICES

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS

sr0 11:0 1 558K 0 rom

vda 252:0 0 10G 0 disk

├─vda1 252:1 0 1M 0 part

├─vda2 252:2 0 200M 0 part /boot/efi

├─vda3 252:3 0 500M 0 part /boot

└─vda4 252:4 0 9.3G 0 part /

vdb 252:16 0 5G 0 disk

vdc 252:32 0 5G 0 disk

vdd 252:48 0 5G 0 disk

#####################################################

FILESYSTEM FREE SPACE STATUS

Filesystem Size Used Avail Use% Mounted on

devtmpfs 844M 0 844M 0% /dev

tmpfs 888M 0 888M 0% /dev/shm

tmpfs 355M 9.4M 346M 3% /run

/dev/vda4 9.4G 1.7G 7.7G 18% /

/dev/vda3 495M 161M 335M 33% /boot

/dev/vda2 200M 7.6M 193M 4% /boot/efi

tmpfs 178M 0 178M 0% /run/user/1000

#####################################################

1. Remove the exercise files and return to the workstation machine.

1. Delete the firstscript.sh and output.txt files.

[student@servera ~]$ rm firstscript.sh output.txt

1. Return to the workstation machine as the student user.

[student@servera ~]$ exit

logout

Connection to servera closed.

[student@workstation ~]$

\*\*Finish\*\*

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ lab finish console-write

This concludes the section.

## Loops and Conditional Constructs in Scripts

```

Objectives

Run repetitive tasks with for loops, evaluate exit codes from commands and scripts, run tests with operators, and create conditional structures with if statements.

Use Loops to Iterate Commands

System administrators often encounter repetitive tasks in their daily activities. A repetitive task example is running a command multiple times on a target, such as checking a process every minute for 10 minutes to know whether it has completed. Another example is running a command once each for multiple targets, such as backing up many databases on a system. The for loop is a Bash looping construct to use for task iterations.

Process Items from the Command Line

In Bash, the for loop construct uses the following syntax:

for VARIABLE in LIST; do

COMMAND VARIABLE

done

The loop processes the strings that you provide in LIST and exits after processing the last string in the list. The for loop temporarily stores each list string as the value of VARIABLE, and then executes the block of commands that use the variable. The variable name is arbitrary. Typically, you reference the variable value with commands in the command block.

Provide the list of strings for the for loop from a list that the user enters directly, or that is generated from shell expansion, such as variable, brace, or file name expansion, or command substitution.

These examples demonstrate different ways to provide strings to for loops:

[user@host ~]$ for HOST in host1 host2 host3; do echo $HOST; done

host1

host2

host3

[user@host ~]$ for HOST in host{1,2,3}; do echo $HOST; done

host1

host2

host3

[user@host ~]$ for HOST in host{1..3}; do echo $HOST; done

host1

host2

host3

[user@host ~]$ for FILE in file{a..c}; do ls $FILE; done

filea

fileb

filec

[user@host ~]$ for PACKAGE in $(rpm -qa | grep kernel); \

do echo "$PACKAGE was installed on \

$(date -d @$(rpm -q --qf "%{INSTALLTIME}\n" $PACKAGE))"; done

kernel-tools-libs-5.14.0-70.2.1.el9\_0.x86\_64 was installed on Thu Mar 24 10:52:40 PM EDT 2022

kernel-tools-5.14.0-70.2.1.el9\_0.x86\_64 was installed on Thu Mar 24 10:52:40 PM EDT 2022

kernel-core-5.14.0-70.2.1.el9\_0.x86\_64 was installed on Thu Mar 24 10:52:46 PM EDT 2022

kernel-modules-5.14.0-70.2.1.el9\_0.x86\_64 was installed on Thu Mar 24 10:52:47 PM EDT 2022

kernel-5.14.0-70.2.1.el9\_0.x86\_64 was installed on Thu Mar 24 10:53:04 PM EDT 2022

[user@host ~]$ for EVEN in $(seq 2 2 10); do echo "$EVEN"; done

2

4

6

8

10

Bash Script Exit Codes

After a script interprets and processes all of its content, the script process exits and passes back control to the parent process that called it. However, a script can be exited before it finishes, such as when the script encounters an error condition. Use the exit command to immediately leave the script, and skip processing the remainder of the script.

Use the exit command with an optional integer argument between 0 and 255, which represents an exit code. An exit code is returned to a parent process to indicate the status at exit. An exit code value of 0 represents a successful script completion with no errors. All other nonzero values indicate an error exit code. The script programmer defines these codes. Use unique values to represent the different error conditions that are encountered. Retrieve the exit code of the last completed command from the built-in $? variable, as in the following examples:

[user@host bin]$ cat hello

#!/usr/bin/bash

echo "Hello, world"

exit 0

[user@host bin]$ ./hello

Hello, world

[user@host bin]$ echo $?

0

When a script's exit command is used without an exit code argument, the script returns the exit code of the last command that was run within the script.

Test Logic for Strings and Directories, and to Compare Values

To ensure that unexpected conditions do not disrupt scripts, it is recommended to verify command input such as command-line arguments, user input, command substitutions, variable expansions, and file name expansions. You can verify integrity in your scripts by using the Bash test command.

All commands produce an exit code on completion.

To see the exit status, view the $? variable immediately after executing the test command. An exit status of 0 indicates a successful exit with nothing to report. Nonzero values indicate some condition or failure. Use various operators to test whether a number is greater than (gt), greater than or equal to (ge), less than (lt), less than or equal to (le), or equal (eq) to another number.

Use operators to test whether a string of text is the same (= or ==) or not the same (!=) as another string of text, or whether the string has zero length (z) or has a non-zero length (n). You can also test whether a regular file (-f) or directory (-d) exists, and has some special attributes, such as if the file is a symbolic link (-L), or if the user has read permissions (-r).

Note

Shell scripting uses many other operator types. The test(1) man page lists the conditional expression operators with descriptions. The bash(1) man page also explains operator use and evaluation, but can be complex to read. Red Hat recommends learning shell scripting through quality books and courses that are dedicated to shell programming.

The following examples demonstrate the test command with Bash numeric comparison operators:

[user@host ~]$ test 1 -gt 0 ; echo $?

0

[user@host ~]$ test 0 -gt 1 ; echo $?

1

Test by using the Bash test command syntax, [ <TESTEXPRESSION> ] or the newer extended test command syntax, [[ <TESTEXPRESSION> ]], which provides features such as file name globbing and regex pattern matching. In most cases, use the [[ <TESTEXPRESSION> ]] syntax.

The following examples demonstrate the Bash test command syntax and numeric comparison operators:

[user@host ~]$ [[ 1 -eq 1 ]]; echo $?

0

[user@host ~]$ [[ 1 -ne 1 ]]; echo $?

1

[user@host ~]$ [[ 8 -gt 2 ]]; echo $?

0

[user@host ~]$ [[ 2 -ge 2 ]]; echo $?

0

[user@host ~]$ [[ 2 -lt 2 ]]; echo $?

1

[user@host ~]$ [[ 1 -lt 2 ]]; echo $?

0

The following examples demonstrate the Bash string comparison operators:

[user@host ~]$ [[ abc = abc ]]; echo $?

0

[user@host ~]$ [[ abc == def ]]; echo $?

1

[user@host ~]$ [[ abc != def ]]; echo $?

0

The following examples demonstrate Bash string unary (one argument) operators:

[user@host ~]$ STRING=''; [[ -z "$STRING" ]]; echo $?

0

[user@host ~]$ STRING='abc'; [[ -n "$STRING" ]]; echo $?

0

Note

The space characters inside the brackets are mandatory, because they separate the words and elements within the test expression. The shell's command parsing routine divides the command elements into words and operators by recognizing spaces and other metacharacters, according to built-in parsing rules. For full treatment of this advanced concept, see the getopt(3) man page. The left square bracket character ([) is itself a built-in alias for the test command. Shell words, whether they are commands, subcommands, options, arguments, or other token elements, are always delimited by spaces.

Conditional Structures

Simple shell scripts represent a collection of commands that are executed from beginning to end. Programmers incorporate decision-making into shell scripts by using conditional structures. A script can execute specific routines when stated conditions are met.

Use the If/Then Construct

The simplest conditional structure is the if/then construct, with the following syntax:

if <CONDITION>; then

<STATEMENT>

...

<STATEMENT>

fi

With this construct, if the script meets the given condition, then it executes the code in the statement block. It does not act if the given condition is not met. Common test conditions in the if/then statements include the previously discussed numeric, string, and file tests. The fi statement at the end closes the if/then construct. The following code section demonstrates an if/then construct to start the psacct service if it is not active:

[user@host ~]$ systemctl is-active psacct > /dev/null 2>&1

[user@host ~]$ if [[ $? -ne 0 ]]; then sudo systemctl start psacct; fi

Use the If/Then/Else Construct

You can further expand the if/then construct to take different sets of actions depending on whether a condition is met. Use the if/then/else construct to accomplish this behavior, as in this example:

if <CONDITION>; then

<STATEMENT>

...

<STATEMENT>

else

<STATEMENT>

...

<STATEMENT>

fi

The following code section demonstrates an if/then/else statement to start the psacct service if it is not active, and to stop it if it is active:

[user@host ~]$ systemctl is-active psacct > /dev/null 2>&1

[user@host ~]$ if [[ $? -ne 0 ]]; then \

sudo systemctl start psacct; \

else \

sudo systemctl stop psacct; \

fi

Use the If/Then/Elif/Then/Else Construct

Expand an if/then/else construct to test more than one condition and to execute a different set of actions when it meets a specific condition. The next example shows the construct for an added condition:

if <CONDITION>; then

<STATEMENT>

...

<STATEMENT>

elif <CONDITION>; then

<STATEMENT>

...

<STATEMENT>

else

<STATEMENT>

...

<STATEMENT>

fi

In this conditional structure, Bash tests the conditions as they are ordered in the script. When a condition is true, Bash executes the actions that are associated with the condition and then skips the remainder of the conditional structure. If none of the conditions are true, then Bash executes the actions in the else clause.

The following example demonstrates an if/then/elif/then/else statement to run the mysql client if the mariadb service is active, or to run the psql client if the postgresql service is active, or to run the sqlite3 client if both the mariadb and the postgresql service are inactive:

[user@host ~]$ systemctl is-active mariadb > /dev/null 2>&1

[user@host ~]$ MARIADB\_ACTIVE=$?

[user@host ~]$ sudo systemctl is-active postgresql > /dev/null 2>&1

[user@host ~]$ POSTGRESQL\_ACTIVE=$?

[user@host ~]$ if [[ "$MARIADB\_ACTIVE" -eq 0 ]]; then \

mysql; \

elif [[ "$POSTGRESQL\_ACTIVE" -eq 0 ]]; then \

psql; \

else \

sqlite3; \

fi

References

bash(1) man page

```

## Guided Exercise: Loops and Conditional Constructs in Scripts

```

In this exercise, you use loops to efficiently print the hostname from multiple servers.

Outcomes

Create a for loop to iterate through a list of items from the command line and in a shell script.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

[student@workstation ~]$ lab start console-commands

Instructions

Use the ssh and hostname commands to print the hostname of the servera and serverb machines to standard output.

[student@workstation ~]$ ssh student@servera hostname

servera.lab.example.com

[student@workstation ~]$ ssh student@serverb hostname

serverb.lab.example.com

Create a for loop to execute the hostname command on the servera and serverb machines.

[student@workstation ~]$ for HOST in servera serverb

do

ssh student@${HOST} hostname

done

servera.lab.example.com

serverb.lab.example.com

Create a shell script in the /home/student/bin directory to execute the same for loop. Ensure that the script is included in the PATH environment variable.

Create the /home/student/bin directory to store the shell script, if the directory does not exist.

[student@workstation ~]$ mkdir ~/bin

Verify that the bin subdirectory of your home directory is in your PATH environment variable.

[student@workstation ~]$ echo $PATH

/home/student/.local/bin:/home/student/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin:/home/student/.venv/labs/bin

Create a shell script called printhostname.sh in the /home/student/bin directory to perform the for loop, and add the following content in the file.

[student@workstation ~]$ vim ~/bin/printhostname.sh

#!/usr/bin/bash

#Execute for loop to print server hostname.

for HOST in servera serverb

do

ssh student@${HOST} hostname

done

exit 0

Give the created script executable permission.

[student@workstation ~]$ chmod +x ~/bin/printhostname.sh

Run the script from your home directory.

[student@workstation ~]$ printhostname.sh

servera.lab.example.com

serverb.lab.example.com

Verify that the exit code of your script is 0.

[student@workstation ~]$ echo $?

0

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ lab finish console-commands

This concludes the section.

```

## Match Text in Command Output with Regular Expressions

```

Objectives

Create regular expressions to match data, apply regular expressions to text files with the grep command, and use grep to search files and data from piped commands.

Write Regular Expressions

Regular expressions provide a pattern matching mechanism to find specific content. The vim, grep, and less commands can use regular expressions. Programming languages such as Perl, Python, and C also support regular expressions, but might differ slightly in syntax.

Regular expressions are a unique language, with their own syntax and rules. This section introduces regular expression syntax as implemented in bash, with examples.

Describe a Simple Regular Expression

The simplest regular expression is an exact match of the string to search. An exact match is when the characters in the regular expression match the type and order of the string.

Imagine that a user is looking through the following file for all occurrences of the pattern cat:

cat

dog

concatenate

dogma

category

educated

boondoggle

vindication

chilidog

The cat string is an exact match of the c character, followed by the a and t characters with no other characters between. Searching the file with the cat string as the regular expression returns the following matches:

cat

concatenate

category

educated

vindication

Match the Start and End of a Line

The regular expression would match the search string anywhere on the line on which it occurred: the beginning, middle, or end of the word or line. Use a line anchor metacharacter to control where on a line to look for a match.

To match only at the beginning of a line, use the caret character (^). To match only at the end of a line, use the dollar sign ($).

With the same file as for the previous example, the ^cat regular expression would match two lines.

cat

category

The cat$ regular expression would find only one match, where the cat characters are at the end of a line.

cat

Locate lines in the file that end with dog, by using an end-of-line anchor to create the dog$ regular expression, which matches two lines:

dog

chilidog

To locate a line that contains only the search expression exactly, use both the beginning and end-of-line anchors. For example, to locate the word cat when it is both at the beginning and the end of a line simultaneously, use ^cat$.

cat

Basic and Extended Regular Expression

The two types of regular expressions are basic regular expressions and extended regular expressions.

One difference between basic and extended regular expressions is in the behavior of the |, +, ?, (, ), {, and } special characters. In basic regular expression syntax, these characters have a special meaning only if they are prefixed with a backslash \ character. In extended regular expression syntax, these characters are special unless they are prefixed with a backslash \ character. Other minor differences apply to how the ^, $, and \* characters are handled.

The grep, sed, and vim commands use basic regular expressions. The grep command -E option, the sed command -E option, and the less command use extended regular expressions.

Wildcard and Multiplier Usage in Regular Expressions

Regular expressions use a dot character (.) as a wildcard to match any single character on a single line. The c.t regular expression searches for a string that contains a c, followed by any single character, followed by a t. Example matches might include cat, concatenate, vindication, cut, and c$t.

With an unrestricted wildcard, you cannot predict the character that matches the wildcard. To match specific characters, replace the unrestricted wildcard with appropriate characters.

The use of bracket characters, such as in the c[aou]t regular expression, matches patterns that start with a c, followed by an a, o, or u, followed by a t. Possible matching expressions can have the cat, cot, and cut strings.

Multipliers are an often used mechanism with wildcards. Multipliers apply to the previous character or wildcard in the regular expression. An often used multiplier is the asterisk (\*) character. When used in a regular expression, the asterisk multiplier matches zero or more occurrences of the multiplied expression. You can use the asterisk with expressions, in addition to characters.

For example, the c[aou]\*t regular expression might match coat or coot. A regular expression of c.\*t matches cat, coat, culvert, and even ct (matching zero characters between the c and the t). Any string that starts with a c, is followed by zero or more characters, and ends with a t must be a match.

Another type of multiplier indicates a more precise number of characters in the pattern. An example of an explicit multiplier is the 'c.\{2\}t' regular expression, which matches any word that begins with a c, followed by exactly any two characters, and ends with a t. The 'c.\{2\}t' expression would match two words in the following example:

cat

coat

convert

cart

covert

cypher

Note

This course introduced two metacharacter text parsing mechanisms: shell pattern matching (also known as file globbing or file name expansion), and regular expressions. Both mechanisms use similar metacharacters, such as the asterisk character (\*), but have differences in metacharacter interpretation and rules.

Pattern matching is a shell technique to specify multiple file names on the command line. Regular expressions represent any form or pattern in text strings, no matter how complex. Regular expressions are internally supported by many text processing commands, such as grep, sed, awk, python, and perl, and in many applications.

Table 1.1. Basic and Extended Regular Expression Syntax

Basic syntax Extended syntax Description

. The period (.) matches any single character.

? The preceding item is optional and is matched at most once.

\* The preceding item is matched zero or more times.

+ The preceding item is matched one or more times.

\{n\} {n} The preceding item is matched exactly n times.

\{n,\} {n,} The preceding item is matched n or more times.

\{,m\} {,m} The preceding item is matched at most m times.

\{n,m\} {n,m} The preceding item is matched at least n times, but not more than m times.

[:alnum:] Alphanumeric characters: [:alpha:] and [:digit:]; in the 'C' locale and ASCII character encoding, this expression is the same as [0-9A-Za-z].

[:alpha:] Alphabetic characters: [:lower:] and [:upper:]; in the 'C' locale and ASCII character encoding, this expression is the same as [A-Za-z].

[:blank:] Blank characters: space and tab.

[:cntrl:] Control characters. In ASCII, these characters have octal codes 000 through 037, and 177 (DEL).

[:digit:] Digits: 0 1 2 3 4 5 6 7 8 9.

[:graph:] Graphical characters: [:alnum:] and [:punct:].

[:lower:] Lowercase letters; in the 'C' locale and ASCII character encoding: a b c d e f g h i j k l m n o p q r s t u v w x y z.

[:print:] Printable characters: [:alnum:], [:punct:], and space.

[:punct:] Punctuation characters; in the 'C' locale and ASCII character encoding: ! " # $ % & ' ( ) \* + , - . / : ; < = > ? @ [ \ ] ^ \_ ' { | } ~.

[:space:] Space characters: in the 'C' locale, it is tab, newline, vertical tab, form feed, carriage return, and space.

[:upper:] Uppercase letters: in the 'C' locale and ASCII character encoding, it is: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z.

[:xdigit:] Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f.

\b Match the empty string at the edge of a word.

\B Match the empty string provided that it is not at the edge of a word.

\< Match the empty string at the beginning of a word.

\> Match the empty string at the end of a word.

\w Match word constituent. Synonym for [\_[:alnum:]].

\W Match non-word constituent. Synonym for [^\_[:alnum:]].

\s Match white space. Synonym for [[:space:]].

\S Match non-white space. Synonym for [^[:space:]].

Match Regular Expressions from the Command Line

The grep command uses regular expressions to isolate matching data. You can use the grep command to match data in a single file or in multiple files. When you use grep to match data in multiple files, it prints the file name followed by a colon character and then the lines that match the regular expression.

Isolating Data with the grep Command

The grep command specifies a regular expression and a file to parse for matches.

[user@host ~]$ grep '^computer' /usr/share/dict/words

computer

computerese

computerise

computerite

computerizable

computerization

computerize

computerized

computerizes

computerizing

computerlike

computernik

computers

Note

It is recommended practice to use single quotation marks to encapsulate the regular expression to protect any shell metacharacters (such as the $, \*, and {} characters). Encapsulating the regular expression ensures that the command and not the shell interprets the characters.

The grep command can process output from other commands by using a pipe operator character (|). The following example shows the grep command parsing lines from the output of another command.

[root@host ~]# ps aux | grep chrony

chrony 662 0.0 0.1 29440 2468 ? S 10:56 0:00 /usr/sbin/chronyd

The grep Command Options

The grep command has many options for controlling how it parses lines.

Table 1.2. Table of Common grep Options

Option Function

-i Use the provided regular expression and do not enforce case sensitivity (run case-insensitive).

-v Display only lines that do not contain matches to the regular expression.

-r Search for data that matches the regular expression recursively in a group of files or directories.

-A NUMBER Display NUMBER of lines after the regular expression match.

-B NUMBER Display NUMBER of lines before the regular expression match.

-e If multiple -e options are used, then multiple regular expressions can be supplied and are used with a logical OR.

-E Use extended regular expression syntax instead of basic regular expression syntax when parsing the provided regular expression.

View the man pages to find other options for the grep command.

Examples of the grep Command

The following examples use various configuration files and log files.

Regular expressions are case-sensitive by default. Use the grep command -i option to run a case-insensitive search. The following example shows an excerpt of the /etc/httpd/conf/httpd.conf configuration file.

[user@host ~]$ cat /etc/httpd/conf/httpd.conf

...output omitted...

ServerRoot "/etc/httpd"

#

# Listen: Allows you to bind Apache to specific IP addresses and/or

# ports, instead of the default. See also the <VirtualHost>

# directive.

#

# Change this to Listen on a specific IP address, but note that if

# httpd.service is enabled to run at boot time, the address may not be

# available when the service starts. See the httpd.service(8) man

# page for more information.

#

#Listen 12.34.56.78:80

Listen 80

...output omitted...

The following example searches for the serverroot regular expression in the /etc/httpd/conf/httpd.conf configuration file.

[user@host ~]$ grep -i serverroot /etc/httpd/conf/httpd.conf

# with "/", the value of ServerRoot is prepended -- so 'log/access\_log'

# with ServerRoot set to '/www' will be interpreted by the

# ServerRoot: The top of the directory tree under which the server's

# ServerRoot at a non-local disk, be sure to specify a local disk on the

# same ServerRoot for multiple httpd daemons, you will need to change at

ServerRoot "/etc/httpd"

Use the grep command -v option to reverse search the regular expression. This option displays only the lines that do not match the regular expression.

In the following example, all lines, regardless of case, that do not contain the server regular expression are returned.

[user@host ~]$ grep -v -i server /etc/hosts

127.0.0.1 localhost.localdomain localhost

172.25.254.254 classroom.example.com classroom

172.25.254.254 content.example.com content

172.25.254.254 materials.example.com materials

### rht-vm-hosts file listing the entries to be appended to /etc/hosts

172.25.250.9 workstation.lab.example.com workstation

172.25.250.254 bastion.lab.example.com bastion

172.25.250.220 utility.lab.example.com utility

172.25.250.220 registry.lab.example.com registry

To view a file without the distraction of comment lines, use the grep command -v option. In the following example, the regular expression matches and excludes all the lines that begin with a hash character (#) or a semicolon (;) character in the /etc/systemd/system/multi-user.target.wants/rsyslog.service file. In that file, the hash character at the beginning of a line indicates a general comment, whereas the semicolon character refers to a commented variable value.

[user@host ~]$ grep -v '^[#;]' \

/etc/systemd/system/multi-user.target.wants/rsyslog.service

[Unit]

Description=System Logging Service

Documentation=man:rsyslogd(8)

Documentation=https://www.rsyslog.com/doc/

[Service]

Type=notify

EnvironmentFile=-/etc/sysconfig/rsyslog

ExecStart=/usr/sbin/rsyslogd -n $SYSLOGD\_OPTIONS

ExecReload=/usr/bin/kill -HUP $MAINPID

UMask=0066

StandardOutput=null

Restart=on-failure

LimitNOFILE=16384

[Install]

WantedBy=multi-user.target

The grep command -e option can search for more than one regular expression at a time. The following example, which uses a combination of the less and grep commands, locates all occurrences of pam\_unix, user root, and Accepted publickey in the /var/log/secure log file.

[root@host ~]# cat /var/log/secure | grep -e 'pam\_unix' \

-e 'user root' -e 'Accepted publickey' | less

Mar 4 03:31:41 localhost passwd[6639]: pam\_unix(passwd:chauthtok): password changed for root

Mar 4 03:32:34 localhost sshd[15556]: Accepted publickey for devops from 10.30.0.167 port 56472 ssh2: RSA SHA256:M8ikhcEDm2tQ95Z0o7ZvufqEixCFCt+wowZLNzNlBT0

Mar 4 03:32:34 localhost systemd[15560]: pam\_unix(systemd-user:session): session opened for user devops(uid=1001) by (uid=0)

To search for text in a file that you opened with the vim or less commands, first enter the slash character (/) and then type the pattern to find. Press Enter to start the search. Press N to find the next match.

[root@host ~]# vim /var/log/boot.log

...output omitted...

[^[[0;32m OK ^[[0m] Finished ^[[0;1;39mdracut pre-pivot and cleanup hook^[[0m.^M

Starting ^[[0;1;39mCleaning Up and Shutting Down Daemons^[[0m...^M

[^[[0;32m OK ^[[0m] Stopped target ^[[0;1;39mRemote Encrypted Volumes^[[0m.^M

[^[[0;32m OK ^[[0m] Stopped target ^[[0;1;39mTimer Units^[[0m.^M

[^[[0;32m OK ^[[0m] Closed ^[[0;1;39mD-Bus System Message Bus Socket^[[0m.^M

/Daemons

[root@host ~]# less /var/log/messages

...output omitted...

Mar 4 03:31:19 localhost kernel: pci 0000:00:02.0: vgaarb: setting as boot VGA device

Mar 4 03:31:19 localhost kernel: pci 0000:00:02.0: vgaarb: VGA device added: decodes=io+mem,owns=io+mem,locks=none

Mar 4 03:31:19 localhost kernel: pci 0000:00:02.0: vgaarb: bridge control possible

Mar 4 03:31:19 localhost kernel: vgaarb: loaded

Mar 4 03:31:19 localhost kernel: SCSI subsystem initialized

Mar 4 03:31:19 localhost kernel: ACPI: bus type USB registered

Mar 4 03:31:19 localhost kernel: usbcore: registered new interface driver usbfs

Mar 4 03:31:19 localhost kernel: usbcore: registered new interface driver hub

Mar 4 03:31:19 localhost kernel: usbcore: registered new device driver usb

/device

References

regex(7) and grep(1) man pages

```

### Guided Exercise: Match Text in Command Output with Regular Expressions

```

In this lab, you search for text in the system logs and the output of commands to find information more efficiently.

Outcomes

Efficiently search for text in log files and configuration files.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that all required resources are available.

[student@workstation ~]$ lab start console-regex

Instructions

Log in to the servera machine as the student user and switch to the root user.

[student@workstation ~]$ ssh student@servera

...output omitted...

[student@servera ~]$ sudo -i

[sudo] password for student: student

[root@servera ~]#

Use the grep command to find the GID and UID for the postfix and postdrop groups and users. To do so, use the rpm -q --scripts command, which queries the information for a specific package and shows the scripts that are used as part of the installation process.

[root@servera ~]# rpm -q --scripts postfix | grep -e 'user' -e 'group'

# Add user and groups if necessary

/usr/sbin/groupadd -g 90 -r postdrop 2>/dev/null

/usr/sbin/groupadd -g 89 -r postfix 2>/dev/null

/usr/sbin/groupadd -g 12 -r mail 2>/dev/null

/usr/sbin/useradd -d /var/spool/postfix -s /sbin/nologin -g postfix -G mail -M -r -u 89 postfix 2>/dev/null

setgid\_group=postdrop \

Modify the previous regular expression to display the first two messages in the /var/log/maillog file. In this search, you do not need to use the caret character (^), because you are not searching for the first character in a line.

[root@servera ~]# grep 'postfix' /var/log/maillog | head -n 2

Apr 1 15:27:16 servera postfix/postfix-script[3121]: starting the Postfix mail system

Apr 1 15:27:16 servera postfix/master[3123]: daemon started -- version 3.5.9, configuration /etc/postfix

Find the name of the queue directory for the Postfix server. Search the /etc/postfix/main.cf configuration file for all information about queues. Use the grep command -i option to ignore case distinctions.

[root@servera ~]# grep -i 'queue' /etc/postfix/main.cf

# testing. When soft\_bounce is enabled, mail will remain queued that

# The queue\_directory specifies the location of the Postfix queue.

queue\_directory = /var/spool/postfix

# QUEUE AND PROCESS OWNERSHIP

# The mail\_owner parameter specifies the owner of the Postfix queue

# is the Sendmail-compatible mail queue listing command.

# setgid\_group: The group for mail submission and queue management

Confirm that the postfix service writes messages to the /var/log/messages file. Use the less command and then the slash character (/) to search the file. Press n to move to the next entry that matches the search. Press q to quit the less command.

[root@servera ~]# less /var/log/messages

...output omitted...

Apr 1 15:27:15 servera systemd[1]: Starting Postfix Mail Transport Agent...

...output omitted...

Apr 1 15:27:16 servera systemd[1]: Started Postfix Mail Transport Agent.

...output omitted...

/Postfix

Use the ps aux command to confirm that the postfix server is currently running. Use the grep command to limit the output to the necessary lines.

[root@servera ~]# ps aux | grep postfix

root 3123 0.0 0.2 38172 4384 ? Ss 15:27 0:00 /usr/libexec/postfix/master -w

postfix 3124 0.0 0.4 45208 8236 ? S 15:27 0:00 pickup -l -t unix -u

postfix 3125 0.0 0.4 45252 8400 ? S 15:27 0:00 qmgr -l -t unix -u

root 3228 0.0 0.1 221668 2288 pts/0 S+ 15:55 0:00 grep --color=auto postfix

Confirm that the qmgr, cleanup, and pickup queues are correctly configured. Use the grep command -e option to match multiple entries in the same file. The /etc/postfix/master.cf file is the configuration file.

[root@servera ~]# grep -e qmgr -e pickup -e cleanup /etc/postfix/master.cf

pickup unix n - n 60 1 pickup

cleanup unix n - n - 0 cleanup

qmgr unix n - n 300 1 qmgr

#qmgr unix n - n 300 1 oqmgr

Return to the workstation machine as the student user.

[root@servera ~]# exit

logout

[student@servera ~]$ exit

logout

Connection to servera closed.

[student@workstation ~]$

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ lab finish console-regex

This concludes the section.

```

### Lab: Improve Command-line Productivity

```

In this lab, you create a Bash script that can filter and get relevant information from different hosts.

Outcomes

Create a Bash script and redirect its output to a file.

Use loops to simplify your code.

Filter the relevant content by using grep and regular expressions.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

[student@workstation ~]$ lab start console-review

Instructions

Create the executable /home/student/bin/bash-lab script file on the workstation machine. The initial content in the script must use the shebang interpreter directive.

On the workstation machine, create the /home/student/bin/ directory if needed.

[student@workstation ~]$ mkdir -p /home/student/bin

Use the vim command to create and edit the /home/student/bin/bash-lab script file.

[student@workstation ~]$ vim ~/bin/bash-lab

Insert the following text and save the file.

#!/usr/bin/bash

Make your script file executable.

[student@workstation ~]$ chmod a+x ~/bin/bash-lab

Edit your newly created script file to store the following information from the servera and serverb machines on the workstation machine. The systems use SSH keys for authentication, and therefore you do not require a password. Store the output of the listed commands from the following table in the /home/student/output-servera and /home/student/output-serverb files respectively on the workstation machine. Use the hash sign (#) for differentiating the output of the successive commands in the output file.

Command or file Content requested

hostname -f Store the entire output.

echo "#####" Append the hash signs to differentiate the following command.

lscpu Get only the lines that start with the CPU string.

echo "#####" Append the hash signs to differentiate the following command.

/etc/selinux/config Ignore empty lines. Also, ignore lines that start with the # character.

echo "#####" Append the hash signs to differentiate the following command.

/var/log/secure Get all "Failed password" entries.

echo "#####" Append the hash signs to differentiate the following command.

Save the required information to the output-servera and output-serverb files in the /﻿home/student directory on workstation.

Note

You can use the sudo command without requiring a password on the servera and serverb hosts. Remember to use a loop to simplify your script. You can also use multiple grep commands that are concatenated with the use of the pipe character (|).

Use the vim command to open and edit the /home/student/bin/bash-lab script file.

[student@workstation ~]$ vim ~/bin/bash-lab

Append the following lines to the /home/student/bin/bash-lab script file. The number of hash signs is arbitrary.

Note

The following output is an example of how you can achieve the requested script. In Bash scripting, you can take different approaches and obtain the same result.

#!/usr/bin/bash

USR='student'

OUT='/home/student/output'

#

for SRV in servera serverb; do

ssh ${USR}@${SRV} "hostname -f" > ${OUT}-${SRV}

echo "#####" >> ${OUT}-${SRV}

ssh ${USR}@${SRV} "lscpu | grep '^CPU'" >> ${OUT}-${SRV}

echo "#####" >> ${OUT}-${SRV}

ssh ${USR}@${SRV} "grep -v '^$' /etc/selinux/config|grep -v '^#'" >> ${OUT}-${SRV}

echo "#####" >> ${OUT}-${SRV}

ssh ${USR}@${SRV} "sudo grep 'Failed password' /var/log/secure" >> ${OUT}-${SRV}

echo "#####" >> ${OUT}-${SRV}

done

Execute the /home/student/bin/bash-lab script, and review the output content on workstation.

On workstation, execute the /home/student/bin/bash-lab script.

[student@workstation ~]$ bash-lab

Review the content of the /home/student/output-servera and /home/student/output-serverb files.

[student@workstation ~]$ cat /home/student/output-servera

servera.lab.example.com

#####

CPU op-mode(s): 32-bit, 64-bit

CPU(s): 2

CPU family: 6

#####

SELINUX=enforcing

SELINUXTYPE=targeted

#####

Apr 1 05:42:07 servera sshd[1275]: Failed password for invalid user operator1 from 172.25.250.9 port 42460 ssh2

Apr 1 05:42:09 servera sshd[1277]: Failed password for invalid user sysadmin1 from 172.25.250.9 port 42462 ssh2

Apr 1 05:42:11 servera sshd[1279]: Failed password for invalid user manager1 from 172.25.250.9 port 42464 ssh2

#####

[student@workstation ~]$ cat /home/student/output-serverb

serverb.lab.example.com

#####

CPU op-mode(s): 32-bit, 64-bit

CPU(s): 2

CPU family: 6

#####

SELINUX=enforcing

SELINUXTYPE=targeted

#####

Apr 1 05:42:14 serverb sshd[1252]: Failed password for invalid user operator1 from 172.25.250.9 port 53494 ssh2

Apr 1 05:42:17 serverb sshd[1257]: Failed password for invalid user sysadmin1 from 172.25.250.9 port 53496 ssh2

Apr 1 05:42:19 serverb sshd[1259]: Failed password for invalid user manager1 from 172.25.250.9 port 53498 ssh2

#####

Evaluation

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ lab grade console-review

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ lab finish console-review

This concludes the section.

```

### Summary

\* Create and execute Bash scripts to accomplish administration tasks.

\*

\* Use loops to iterate through a list of items from the command line and in a shell script.

\*

\* Use conditional structures to incorporate decision-making into shell scripts.

\*

\* Search for text in log and configuration files by using regular expressions and the grep command.

## Chapter 2. Schedule Future Tasks

```

Schedule a Deferred User Job

Guided Exercise: Schedule a Deferred User Job

Schedule Recurring User Jobs

Guided Exercise: Schedule Recurring User Jobs

Schedule Recurring System Jobs

Guided Exercise: Schedule Recurring System Jobs

Manage Temporary Files

Guided Exercise: Manage Temporary Files

Quiz: Schedule Future Tasks

Summary

Abstract

Goal Schedule tasks to execute at a specific time and date.

Objectives

Set up a command to run once at a future time.

Schedule commands to run on a repeating schedule with a user's crontab file.

Schedule commands to run on a repeating schedule with the system crontab file and directories.

Enable and disable systemd timers, and configure a timer that manages temporary files.

Sections

Schedule a Deferred User Job (and Guided Exercise)

Schedule Recurring User Jobs (and Guided Exercise)

Schedule Recurring System Jobs (and Guided Exercise)

Manage Temporary Files (and Guided Exercise)

Schedule Future Tasks (Quiz)

Schedule a Deferred User Job

Objectives

Set up a command to run once at a future time.

Describe Deferred User Tasks

Sometimes you might need to run one or more commands at a specific future time. An example is a user who schedules a long-running maintenance task to occur in the middle of the night. Another example is a system administrator who is working on a firewall configuration and queues a safety job to reset the firewall settings to a former working state in ten minutes' time. The system administrator then deactivates the job before it runs, unless the new firewall configuration worked.

These scheduled commands are called tasks or jobs, and the deferred term indicates that these tasks run in the future.

One available solution for Red Hat Enterprise Linux users to schedule deferred tasks is the at command, which is installed and enabled by default. The at package provides the atd system daemon and the at and atq commands to interact with the daemon.

Any user can queue jobs for the atd daemon by using the at command. The atd daemon provides 26 queues, identified from a to z, where jobs in alphabetically later queues get lower system priority (with higher nice values, as discussed in a later chapter).

Schedule Deferred User Tasks

Use the at TIMESPEC command to start entering a new job to schedule. The at command then reads from STDIN (your keyboard) to obtain the commands to run. When manually entering commands, complete the input by pressing Ctrl+D on an empty line. You can use input redirection from a script file for entering more complex commands. For example, use the at now +5min < myscript command to schedule the commands in myscript to start in 5 minutes, without needing to type the commands manually in a terminal window.

The at command TIMESPEC argument accepts natural time specifications to describe when a job should run. For example, specify a time as 02:00pm, 15:59, midnight, or even teatime, followed by an optional date or number of days in the future.

The TIMESPEC argument expects time and date specifications in that order. If you provide the date and not the time, then the time defaults to the current time. If you provide the time and not the date, then the date is considered to be matched, and the jobs run when the time next matches.

The following example shows a job schedule without providing the date. The at command schedules the job for today or tomorrow depending whether the time has passed.

[user@host ~]$ date

Wed May 18 21:01:18 CDT 2022

[user@host ~]$ at 21:03 < myscript

job 3 at Wed May 18 21:03:00 2022

[user@host ~]$ at 21:00 < myscript

job 4 at Thu May 19 21:00:00 2022

The man pages for the at command and other documentation sources use lowercase to write the natural time specifications. You can use lowercase, sentence case, or uppercase. Here are examples of time specifications that you can use:

now +5min

teatime tomorrow (teatime is 16:00)

noon +4 days

5pm august 3 2021

For other valid time specifications, refer to the local timespec document listed in the references.

Inspect and Manage Deferred User Jobs

For an overview of the pending jobs for the current user, use the atq or the at -l command.

[user@host ~]$ atq

28 Mon May 16 05:13:00 2022 a user

29 Tue May 17 16:00:00 2022 h user

30 Wed May 18 12:00:00 2022 a user

In the preceding output, every line represents a different scheduled future job. The following description applies to the first line of the output:

28 is the unique job number.

Mon May 16 05:13:00 2022 is the execution date and time for the scheduled job.

a indicates that the job is scheduled with the default queue a.

user is the owner of the job (and the user that the job runs as).

Important

Unprivileged users can view and manage only their own jobs. The root user can view and manage all jobs.

Use the at -c JOBNUMBER command to inspect the commands that run when the atd daemon executes a job. This command shows the job's environment, which is set from the user's environment when they created the job, and the command syntax to run.

Remove Jobs from Schedule

The atrm JOBNUMBER command removes a scheduled job. Remove the scheduled job when you no longer need it, for example, when a remote firewall configuration succeeded, and you do not need to reset it.

References

at(1) and atd(8) man pages

/usr/share/doc/at/timespec

```

### Guided Exercise: Schedule a Deferred User Job

```

In this exercise, you use the at command to schedule several commands to run at specified future times.

Outcomes

Schedule a job to run at a specified future time.

Inspect the commands that a scheduled job runs.

Delete the scheduled jobs.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

[student@workstation ~]$ lab start scheduling-at

Instructions

From workstation, open an SSH session to servera as the student user.

[student@workstation ~]$ ssh student@servera

...output omitted...

[student@servera ~]$

Schedule a job to run in two minutes from now. Save the output of the date command to the /home/student/myjob.txt file.

Pass the date >> /home/student/myjob.txt string as the input to the at command, so that the job runs in two minutes from now.

[student@servera ~]$ echo "date >> /home/student/myjob.txt" | at now +2min

warning: commands will be executed using /bin/sh

job 1 at Thu Feb 16 18:51:16 2023

List the scheduled jobs.

[student@servera ~]$ atq

1 Thu Feb 16 18:51:16 2023 a student

Monitor the deferred jobs queue in real time. After the atd daemon executes, it removes the job from the queue.

The command updates the output of the atq command every two seconds, by default. After the atd daemon removes the deferred job from the queue, press Ctrl+c to exit the watch command and return to the shell prompt.

[student@servera ~]$ watch atq

Every 2.0s: atq servera.lab.example.com: Thu Feb 16 17:58:50 2023

1 Thu Feb 16 18:51:16 2023 a student

Verify that the contents of the /home/student/myjob.txt file match the output of the date command.

The output matches the output of the date command, which confirms that the scheduled job executed successfully.

[student@servera ~]$ cat myjob.txt

Thu Feb 16 06:51:16 PM EDT 2023

Interactively schedule a job in the g queue that runs at teatime (16:00). The job should print the It's teatime message to the /home/student/tea.txt file. Append the new messages to the /home/student/tea.txt file.

[student@servera ~]$ at -q g teatime

warning: commands will be executed using /bin/sh

at> echo "It's teatime" >> /home/student/tea.txt

at> Ctrl+d

job 2 at Fri Feb 17 16:00:00 2023

Interactively schedule another job with the b queue that runs at 16:05. The job should print The cookies are good message to the /home/student/cookies.txt file. Append the new messages to the /home/student/cookies.txt file.

[student@servera ~]$ at -q b 16:05

warning: commands will be executed using /bin/sh

at> echo "The cookies are good" >> /home/student/cookies.txt

at> Ctrl+d

job 3 at Fri Feb 17 16:05:00 2023

Inspect the commands in the pending jobs.

View the job numbers of the pending jobs.

Note the job numbers in the output, which might vary on your system. Use the job numbers from your system.

[student@servera ~]$ atq

2 Fri Feb 17 16:00:00 2023 g student

3 Fri Feb 17 16:05:00 2023 b student

View the commands in the pending job number 2. Replace the job number if it changed for you.

The job executes an echo command that appends the It's teatime message to the /home/student/tea.txt file.

[student@servera ~]$ at -c 2

...output omitted...

echo "It's teatime" >> /home/student/tea.txt

marcinDELIMITER1d7be6a7

View the commands in the pending job number 3. Replace the job number if it changed for you.

The job executes an echo command that appends the message The cookies are good to the /home/student/cookies.txt file.

[student@servera ~]$ at -c 3

...output omitted...

echo "The cookies are good" >> /home/student/cookies.txt

marcinDELIMITER44662c6f

View the job number of a job that runs at teatime (16:00), and remove it by using the atrm command.

[student@servera ~]$ atq

2 Fri Feb 17 16:00:00 2023 g student

3 Fri Feb 17 16:05:00 2023 b student

[student@servera ~]$ atrm 2

Verify that the scheduled job to run at teatime (16:00) no longer exists.

View the list of pending jobs, and confirm that the scheduled job to run at teatime (16:00) no longer exists.

[student@servera ~]$ atq

3 Fri Feb 17 16:05:00 2023 b student

Return to the workstation machine as the student user.

[student@servera ~]$ exit

logout

Connection to servera closed.

[student@workstation ~]$

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ lab finish scheduling-at

This concludes the section.

```

### Schedule Recurring User Jobs

```

Objectives

Schedule commands to run on a repeating schedule with a user's crontab file.

Describe Recurring User Jobs

Recurring jobs are scheduled to run repeatedly. Red Hat Enterprise Linux systems provide the crond daemon, which is enabled and started by default. The crond daemon reads multiple configuration files: one per user, and a set of system-wide files. Each user has a personal file that they edit with the crontab -e command. When executing recurring jobs, these configuration files provide detailed control to users and administrators. If the scheduled job is not written to use redirection, then the crond daemon emails any generated output or errors to the job owner.

Schedule Recurring User Jobs

Use the crontab command to manage scheduled jobs. The following list shows the commands that a local user can use to manage their jobs:

Table 2.1. Examples Of The crontab Command

Command Intended use

crontab -l List the jobs for the current user.

crontab -r Remove all jobs for the current user.

crontab -e Edit jobs for the current user.

crontab filename Remove all jobs, and replace them with jobs that are read from filename. This command uses stdin input when no file is specified.

A privileged user might use the crontab command -u option to manage jobs for another user. The crontab command is never used to manage system jobs, and using the crontab commands as the root user is not recommended due to the ability to exploit personal jobs that are configured to run as root. Configure such privileged jobs as described in the later section that describes recurring system jobs.

Describe User Job Format

The crontab -e command invokes the vim editor by default unless the EDITOR environment variable is set for another editor. Each job must use a unique line in the crontab file. Follow these recommendations for valid entries when writing recurring jobs:

Empty lines for ease of reading

Comments on lines that start with the number sign (#)

Environment variables with a NAME=value format, which affects all lines after the line where they are declared

Standard variable settings include the SHELL variable, to declare the shell that is used for interpreting the remaining lines of the crontab file. The MAILTO variable determines who should receive the emailed output.

Note

The ability to send an email requires additional system configuration for a local mail server or an SMTP relay.

The fields in the crontab file appear in the following order:

Minutes

Hours

Day of month

Month

Day of week

Command

The command executes when the Day of month or Day of week fields use the same value other than the \* character. For example, to run a command on the 11th day of every month, and every Friday at 12:15 (24-hour format), use the following job format:

15 12 11 \* Fri command

The first five fields all use the same syntax rules:

Use the \* character to execute in every possible instance of the field.

A number to specify the number of minutes or hours, a date, or a day of the week. For days of the week, 0 equals Sunday, 1 equals Monday, 2 equals Tuesday, and so on. 7 also equals Sunday.

Use x-y for a range, which includes the x and y values.

Use x,y for lists. Lists might include ranges as well, for example, 5,10-13,17 in the Minutes column, for a job to run at 5, 10, 11, 12, 13, and 17 minutes past the hour.

The \*/x indicates an interval of x; for example, \*/7 in the Minutes column runs a job every seven minutes.

Additionally, 3-letter English abbreviations are used for months or days of the week, for example, Jan, Feb, and Mon, Tue.

The last field contains the full command with options and arguments to execute with the default shell. If the command contains an unescaped percentage sign (%), then that percentage sign is treated as a newline character, and everything after the percentage sign passes to the command as stdin input.

Examples of Recurring User Jobs

The following job executes the /usr/local/bin/yearly\_backup command at exactly 09:00 on 3 February, every year. February is represented as the number 2 in the example, because it is the second month of the year.

0 9 3 2 \* /usr/local/bin/yearly\_backup

The following job sends an email that contains the Chime word to the owner of this job every five minutes between and including 09:00 and 16:00, but only on each Friday in July.

\*/5 9-16 \* Jul 5 echo "Chime"

The preceding 9-16 range of hours means that the job timer starts at the ninth hour (09:00) and continues until the end of the sixteenth hour (16:59). The job starts executing at 09:00 with the last execution at 16:55, because five minutes after 16:55 is 17:00, which is beyond the given scope of hours.

If a range is specific for the hours instead of a single value, then all hours within the range will match. Therefore, with the hours of 9-16, this example matches every five minutes from 09:00 through 16:55.

Note

This example job sends the output as an email, because crond recognizes that the job allowed output to go to the STDIO channel without redirection. Because cron jobs run in a background environment without an output device (known as a controlling terminal), crond buffers the output and creates an email to send it to the specified user in the configuration. For system jobs, the email is sent to the root account.

The following job runs the /usr/local/bin/daily\_report command every working day (Monday to Friday) two minutes before midnight.

58 23 \* \* 1-5 /usr/local/bin/daily\_report

The following job executes the mutt command to send the Checking in mail message to the developer@example.com recipient every working day (Monday to Friday), at 9 AM.

0 9 \* \* 1-5 mutt -s "Checking in" developer@example.com % Hi there, just checking in.

References

crond(8), crontab(1), and crontab(5) man pages

```

### Guided Exercise: Schedule Recurring User Jobs

```

In this exercise, you schedule commands to run on a repeating schedule as a non-privileged user, with the crontab command.

Outcomes

Schedule recurring jobs to run as a non-privileged user.

Inspect the commands that a scheduled recurring job runs.

Remove scheduled recurring jobs.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

[student@workstation ~]$ lab start scheduling-cron

Instructions

Log in to the servera machine as the student user .

[student@workstation ~]$ ssh student@servera

...output omitted...

[student@servera ~]$

Schedule a recurring job as the student user that appends the current date and time to the /home/student/my\_first\_cron\_job.txt file every two minutes. Use the date command to display the current date and time. The job must run only from one day before to one day after the current time. The job must not run on any other day.

Use the date command to display the current date and time. Note the day of the week, which you need for the next steps.

[student@workstation ~]$ date

Wed Mar 15 07:33:01 PM EDT 2023

[student@servera ~]$

Note

You can use the date -d "last day" +%a command to display the day before the current time, and the date -d "next day" +%a command to display the day after the current time.

[student@servera ~]$ date -d "last day" +%a

Tue

[student@servera ~]$ date -d "next day" +%a

Thu

Open the crontab file with the default text editor.

[student@servera ~]$ crontab -e

Insert the following line. Replace the range of days from one day before to one day after the current time:

\*/2 \* \* \* Tue-Thu /usr/bin/date >> /home/student/my\_first\_cron\_job.txt

Press Esc and type :wq to save the changes and exit the editor. When the editor exits, you should see the following output:

...output omitted...

crontab: installing new crontab

[student@servera ~]$

Use the crontab -l command to list the scheduled recurring jobs. Inspect the command that you scheduled to run as a recurring job in the preceding step.

Verify that the job runs the /usr/bin/date command and appends its output to the /﻿home/student/my\_first\_cron\_job.txt file.

[student@servera ~]$ crontab -l

\*/2 \* \* \* Tue-Thu /usr/bin/date >> /home/student/my\_first\_cron\_job.txt

Instruct your shell prompt to sleep until the /home/student/my\_first\_cron\_job.txt file is created because of the successful execution of the recurring job that you scheduled. Wait for your shell prompt to return.

The while command uses ! test -f to continue to run a loop, and sleeps for one second until the my\_first\_cron\_job.txt file is created in the /home/student directory.

[student@servera ~]$ while ! test -f my\_first\_cron\_job.txt; do sleep 1s; done

Verify that the contents of the /home/student/my\_first\_cron\_job.txt file match the output of the date command.

[student@servera ~]$ cat my\_first\_cron\_job.txt

Wed Mar 15 07:40:01 PM EDT 2023

Remove all the scheduled recurring jobs for the student user.

Remove all the scheduled recurring jobs for the student user.

[student@servera ~]$ crontab -r

Verify that no recurring jobs exist for the student user.

[student@servera ~]$ crontab -l

no crontab for student

Return to the workstation machine as the student user.

[student@servera ~]$ exit

logout

Connection to servera closed.

[student@workstation ~]$

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ lab finish scheduling-cron

This concludes the section.

```

### Schedule Recurring System Jobs

```

Objectives

Schedule commands to run on a repeating schedule with the system crontab file and directories.

Recurring System Jobs

System administrators often need to run recurring jobs. It is best to run these jobs from system accounts rather than from user accounts. Schedule these jobs with system-wide crontab files instead of with the crontab command. Job entries in the system-wide crontab files are similar to the users' crontab entries. The system-wide crontab files have an extra field before the command field to specify the user that runs the command.

The /etc/crontab file has a syntax diagram in the comments.

SHELL=/bin/bash

PATH=/sbin:/bin:/usr/sbin:/usr/bin

MAILTO=root

# For details see man 4 crontabs

# Example of job definition:

# .---------------- minute (0 - 59)

# | .------------- hour (0 - 23)

# | | .---------- day of month (1 - 31)

# | | | .------- month (1 - 12) OR jan,feb,mar,apr ...

# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat

# | | | | |

# \* \* \* \* \* user-name command to be executed

The /etc/crontab file and other files in the /etc/cron.d/ directory define the recurring system jobs. Always create custom crontab files in the /etc/cron.d/ directory to schedule recurring system jobs. Place the custom crontab file in the /etc/cron.d directory to prevent a package update from overwriting the /etc/crontab file. Packages that require recurring system jobs place their crontab files in the /etc/cron.d/ directory with the job entries. Administrators also use this location to group related jobs into a single file.

The crontab system also includes repositories for scripts to run every hour, day, week, and month. These repositories are placed in the /etc/cron.hourly/, /etc/cron.daily/, /etc/cron.weekly/, and /etc/cron.monthly/ directories. These directories contain executable shell scripts, not crontab files.

Note

Use the chmod +x script\_name command to make a script executable. A script must be executable to run.

Run Periodic Commands with Anacron

The run-parts command also runs the daily, weekly, and monthly jobs from the /etc/anacrontab configuration file.

The /etc/anacrontab file ensures that scheduled jobs always run and are not skipped accidentally because the system was turned off or hibernated. For example, when a system job that runs daily was not executed at a specified time because the system was rebooting, then the job is completed when the system becomes ready. A delay might occur before the job starts, if specified in the Delay in minutes parameter in the /etc/anacrontab file.

Files in the /var/spool/anacron/ directory determine the daily, weekly, and monthly jobs. When the crond daemon starts a job from the /etc/anacrontab file, it updates the timestamps of those files. With this timestamp, you can determine the last time that the job executed. The syntax of the /etc/anacrontab file is different from the regular crontab configuration files. The /etc/anacrontab file contains four fields per line, as follows.

Period in days

Defines the interval in days for the job to run on a recurring schedule. This field accepts an integer or a macro value. For example, the macro @daily is equivalent to the 1 integer, which executes the job daily. Similarly, the macro @weekly is equivalent to the 7 integer, which executes the job weekly.

Delay in minutes

Defines the time that the crond daemon must wait before it starts the job.

Job identifier

Identifies the unique name of the job in the log messages.

Command

The command to be executed.

The /etc/anacrontab file also contains environment variable declarations with the NAME=value syntax. The START\_HOURS\_RANGE variable specifies the time interval for the jobs to run. Jobs do not start outside this range. When a job does not run within this time interval on a particular day, then the job must wait until the next day for execution.

Systemd Timer

The systemd timer unit activates another unit of a different type (such as a service), whose unit name matches the timer unit name. The timer unit allows timer-based activation of other units. The systemd timer unit logs timer events in system journals for easier debugging.

Sample Timer Unit

The sysstat package provides the systemd timer unit, called the sysstat-collect.timer service, to collect system statistics every 10 minutes. The following output shows the contents of the /usr/lib/systemd/system/sysstat-collect.timer configuration file.

...output omitted...

[Unit]

Description=Run system activity accounting tool every 10 minutes

[Timer]

OnCalendar=\*:00/10

[Install]

WantedBy=sysstat.service

The OnCalendar=\*:00/10 option signifies that this timer unit activates the corresponding sysstat-collect.service unit every 10 minutes. You might specify more complex time intervals.

For example, a 2022-04-\* 12:35,37,39:16 value against the OnCalendar option causes the timer unit to activate the corresponding service unit at the 12:35:16, 12:37:16, and 12:39:16 times, every day during April 2022. You might also specify relative timers with the OnUnitActiveSec option. For example, with the OnUnitActiveSec=15min option, the timer unit triggers the corresponding unit to start 15 minutes after the last time that the timer unit activated its corresponding unit.

Important

Do not modify any units in the configuration files under the /usr/lib/systemd/system directory, because the systemd unit overrides the configuration changes in that file. Create a copy of the configuration file in the /etc/systemd/system directory, and then modify the copied file to prevent any update to the provider package from overriding the changes. If two files exist with the same name in the /﻿usr/lib/systemd/system and /etc/systemd/system directories, then the systemd timer unit parses the file in the /etc/systemd/system directory.

After you change the timer unit configuration file, use the systemctl daemon-reload command to ensure that the systemd timer unit loads the changes.

[root@host ~]# systemctl daemon-reload

After reloading the systemd daemon configuration, use the systemctl command to activate the timer unit.

[root@host ~]# systemctl enable --now <unitname>.timer

References

crontab(5), anacron(8), anacrontab(5), systemd.time(7), systemd.timer(5), and crond(8) man pages

```

### Guided Exercise: Schedule Recurring System Jobs

```

In this exercise, you schedule commands to run on various schedules by adding configuration files to the system crontab directories.

Outcomes

Schedule a recurring system job to count the number of active users.

Update the systemd timer unit that gathers system activity data.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

[student@workstation ~]$ lab start scheduling-system

Instructions

Log in to the servera machine as the student user and switch to the root user.

[student@workstation ~]$ ssh student@servera

...output omitted...

[student@servera ~]$ sudo -i

[sudo] password for student: student

[root@servera ~]#

Schedule a recurring system job that generates a log message to indicate the number of active users in the system. This job must run daily and use the w -h | wc -l command to retrieve the number of active users in the system. Use the logger command to generate the log message of currently active users.

Create the /etc/cron.daily/usercount script file with the following content:

#!/bin/bash

USERCOUNT=$(w -h | wc -l)

logger "There are currently ${USERCOUNT} active users"

Make the script file executable.

[root@servera ~]# chmod +x /etc/cron.daily/usercount

Install the sysstat package. The timer unit must trigger the service unit every ten minutes to collect system activity data with the /usr/lib64/sa/sa1 shell script. Change the timer unit configuration file to collect the system activity data every two minutes.

Install the sysstat package.

[root@servera ~]# dnf install sysstat

...output omitted...

Is this ok [y/N]: y

...output omitted...

Complete!

Copy the /usr/lib/systemd/system/sysstat-collect.timer file to the /﻿etc/systemd/system/sysstat-collect.timer file.

[root@servera ~]# cp /usr/lib/systemd/system/sysstat-collect.timer \

/etc/systemd/system/sysstat-collect.timer

Edit the /etc/systemd/system/sysstat-collect.timer file for the timer unit to run every two minutes. Replace any occurrence of the 10 minutes string with 2 minutes throughout the unit configuration file, including the occurrences in the commented lines. Use the vim /etc/systemd/system/sysstat-collect.timer command to edit the configuration file.

From these changes, the sysstat-collect.timer unit triggers the sysstat-collect.service unit every two minutes, and collects the system activity data in a binary file in the /var/log/sa directory.

...output omitted...

# Activates activity collector every 2 minutes

[Unit]

Description=Run system activity accounting tool every 2 minutes

[Timer]

OnCalendar=\*:00/2

[Install]

WantedBy=sysstat.service

Notify the systemd daemon of the changes.

[root@servera ~]# systemctl daemon-reload

Activate the sysstat-collect.timer unit.

[root@servera ~]# systemctl enable --now sysstat-collect.timer

...output omitted...

Wait until the binary file is created in the /var/log/sa directory.

The while command, ls /var/log/sa | wc -l returns 0 when the file does not exist, or returns 1 when the file exists. The while command pauses for one second when the file is not present. The while loop exits when the file is present.

[root@servera ~]# while [ $(ls /var/log/sa | wc -l) -eq 0 ]; \

do sleep 1s; done

Verify that the binary file in the /var/log/sa directory was modified within two minutes.

[root@servera ~]# ls -l /var/log/sa

total 4

-rw-r--r--. 1 root root 2540 Apr 5 04:08 sa05

[root@servera ~]# date

Tue Apr 5 04:08:29 AM EDT 2022

Return to the workstation machine as the student user.

[root@servera ~]# exit

logout

[student@servera ~]$ exit

logout

Connection to servera closed.

[student@workstation ~]$

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ lab finish scheduling-system

This concludes the section.

```

### Manage Temporary Files

```

Objectives

Enable and disable systemd timers, and configure a timer that manages temporary files.

Manage Temporary Files

Most critical applications and services use temporary files and directories. Some applications and users use the /tmp directory to hold transient working data, whereas other applications use task-specific locations such as daemon- and user-specific volatile directories under /run, which exist only in memory. When the system reboots or loses power, memory-based file systems are self-cleaning.

Commonly, daemons and scripts operate correctly only when their expected temporary files and directories exist. Additionally, purging temporary files on persistent storage is necessary to prevent disk space issues or stale working data.

Red Hat Enterprise Linux includes the systemd-tmpfiles tool, which provides a structured and configurable method to manage temporary directories and files.

At system boot, one of the first systemd service units to launch is the systemd-tmpfiles-setup service. This service runs the systemd-tmpfiles command --create --remove option, which reads instructions from the /usr/lib/tmpfiles.d/\*.conf, /run/tmpfiles.d/\*.conf, and /etc/tmpfiles.d/\*.conf configuration files. These configuration files list files and directories that the systemd-tmpfiles-setup service is instructed to create, delete, or secure with permissions.

Clean Temporary Files with a Systemd Timer

To prevent long-running systems from filling up their disks with stale data, a systemd timer unit called systemd-tmpfiles-clean.timer triggers at a regular interval the systemd-tmpfiles-clean.service unit, which executes the systemd-tmpfiles --clean command.

A systemd timer unit configuration has a [Timer] section to indicate how to start the service with the same name as the timer.

Use the following systemctl command to view the contents of the systemd-tmpfiles-clean.timer unit configuration file.

[user@host ~]$ systemctl cat systemd-tmpfiles-clean.timer

# /usr/lib/systemd/system/systemd-tmpfiles-clean.timer

# SPDX-License-Identifier: LGPL-2.1-or-later

#

# This file is part of systemd.

#

# systemd is free software; you can redistribute it and/or modify it

# under the terms of the GNU Lesser General Public License as published by

# the Free Software Foundation; either version 2.1 of the License, or

# (at your option) any later version.

[Unit]

Description=Daily Cleanup of Temporary Directories

Documentation=man:tmpfiles.d(5) man:systemd-tmpfiles(8)

ConditionPathExists=!/etc/initrd-release

[Timer]

OnBootSec=15min

OnUnitActiveSec=1d

In the preceding configuration, the OnBootSec=15min parameter indicates that the systemd-tmpfiles-clean.service unit gets triggered 15 minutes after the system boots up. The OnUnitActiveSec=1d parameter indicates that any further trigger to the systemd-tmpfiles-clean.service unit happens 24 hours after the service unit was last activated.

Change the parameters in the systemd-tmpfiles-clean.timer unit configuration file to meet your requirements. For example, a 30min value for the OnUnitActiveSec parameter triggers the systemd-tmpfiles-clean.service unit 30 minutes after the service unit is last activated. As a result, the systemd-tmpfiles-clean.service unit gets triggered every 30 minutes after the changes are recognized.

After you change the timer unit configuration file, use the systemctl daemon-reload command to ensure that the systemd daemon loads the new configuration.

[root@host ~]# systemctl daemon-reload

Clean Temporary Files Manually

The systemd-tmpfiles --clean command parses the same configuration files as the systemd-tmpfiles --create command, but instead of creating files and directories, it purges all files that were not accessed, changed, or modified more recently than the maximum age that is defined in the configuration file.

For detailed information about the format of the configuration files for the systemd-tmpfiles service, see the tmpfiles.d(5) man page. The syntax consists of the following columns: Type, Path, Mode, UID, GID, Age, and Argument. Type refers to the action for the systemd-tmpfiles service to take; for example, d to create a directory if it does not exist, or Z to recursively restore SELinux contexts, file permissions, and ownership.

The following command purges a configuration with explanations:

d /run/systemd/seats 0755 root root -

When you create files and directories, create the /run/systemd/seats directory if it does not exist, with the root user and the root group as owners, and with permissions of rwxr-xr-x. If this directory does exist, then take no action. The systemd-tmpfiles service does not purge this directory automatically.

D /home/student 0700 student student 1d

Create the /home/student directory if it does not exist. If it does exist, then remove all its contents. When the system runs the systemd-tmpfiles --clean command, it removes from the directory all files that you did not access, change, or modify for more than one day.

L /run/fstablink - root root - /etc/fstab

Create the /run/fstablink symbolic link, to point to the /etc/fstab directory. Never automatically purge this line.

Configuration File Precedence

The systemd-tmpfiles-clean service configuration files can exist in three places:

/etc/tmpfiles.d/\*.conf

/run/tmpfiles.d/\*.conf

/usr/lib/tmpfiles.d/\*.conf

Use the files in the /etc/tmpfiles.d/ directory to configure custom temporary locations, and to override vendor-provided defaults. The files in the /run/tmpfiles.d/ directory are volatile files, which normally daemons use to manage their own runtime temporary files. Relevant RPM packages provide the files in the /usr/lib/tmpfiles.d/ directory; therefore do not edit these files.

If a file in the /run/tmpfiles.d/ directory has the same file name as a file in the /usr/lib/tmpfiles.d/ directory, then the service uses the file in the /run/tmpfiles.d/ directory. If a file in the /etc/tmpfiles.d/ directory has the same file name as a file in either the /run/tmpfiles.d/ or the /usr/lib/tmpfiles.d/ directories, then the service uses the file in the /﻿etc/tmpfiles.d/ directory.

Given these precedence rules, you can override vendor-provided settings by copying the relevant file to the /etc/tmpfiles.d/ directory and then editing it. By using these configuration locations correctly, you can manage administrator-configured settings from a central configuration management system, and package updates do not overwrite your configured settings.

Note

When testing new or modified configurations, apply only the commands from a single configuration file at a time. Specify the name of the single configuration file on the systemd-tmpfiles command line.

References

systemd-tmpfiles(8), tmpfiles.d(5), stat(1), stat(2), and systemd.timer(5) man pages

```

### Guided Exercise: Manage Temporary Files

```

In this exercise, you configure systemd-tmpfiles to change how quickly it removes temporary files from the /tmp directory, and also to periodically purge files from another directory.

Outcomes

Configure systemd-tmpfiles to remove unused temporary files from the /tmp directory.

Configure systemd-tmpfiles to periodically purge files from another directory.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

[student@workstation ~]$ lab start scheduling-tempfiles

Instructions

Log in to the servera system as the student user and switch to the root user.

[student@workstation ~]$ ssh student@servera

...output omitted...

[student@servera ~]$ sudo -i

[sudo] password for student: student

[root@servera ~]#

Configure the systemd-tmpfiles service to clean the /tmp directory of any unused files from the last five days. Ensure that a package update does not overwrite the configuration files.

Copy the /usr/lib/tmpfiles.d/tmp.conf file to the /etc/tmpfiles.d directory.

[root@servera ~]# cp /usr/lib/tmpfiles.d/tmp.conf \

/etc/tmpfiles.d/tmp.conf

Search for the configuration line in the /etc/tmpfiles.d/tmp.conf file that applies to the /tmp directory. Replace the existing age of the temporary files in that configuration line with the new age of 5 days. Remove from the file all the other lines, including the commented lines. You can use the vim /etc/tmpfiles.d/tmp.conf command to edit the configuration file.

In the configuration, the q type is the same as the d type, and instructs the systemd-tmpfiles service to create the /tmp directory if it does not exist. The directory's octal permissions must be set to 1777. Both the owning user and group of the /tmp directory must be root. The /tmp directory must not contain the unused temporary files from the last five days.

The /etc/tmpfiles.d/tmp.conf file should appear as follows:

q /tmp 1777 root root 5d

Verify the /etc/tmpfiles.d/tmp.conf file configuration.

Because the command does not return any errors, it confirms that the configuration settings are correct.

[root@servera ~]# systemd-tmpfiles --clean /etc/tmpfiles.d/tmp.conf

Add a new configuration that ensures that the /run/momentary directory exists, and that user and group ownership is set to the root user. The octal permissions for the directory must be 0700. The configuration must purge from this directory any files that remain unused in the last 30 seconds.

Create the /etc/tmpfiles.d/momentary.conf file with the following content.

With the configuration, the systemd-tmpfiles service ensures that the /run/momentary directory exists and that its octal permissions are set to 0700. The ownership of the /run/momentary directory must be the root user and group. The service purges from this directory any file if it remains unused for 30 seconds.

[root@servera ~]# vim /etc/tmpfiles.d/momentary.conf

d /run/momentary 0700 root root 30s

Verify the /etc/tmpfiles.d/momentary.conf file configuration. The command creates the /run/momentary directory if it does not exist.

Because the command does not return any errors, it confirms that the configuration settings are correct.

[root@servera ~]# systemd-tmpfiles --create \

/etc/tmpfiles.d/momentary.conf

Verify that the systemd-tmpfiles command creates the /run/momentary directory with the appropriate permissions, owner, and group owner.

The octal permission for the /run/momentary directory is set to 0700, and the user and group ownership are set to root.

[root@servera ~]# ls -ld /run/momentary

drwx------. 2 root root 40 Apr 4 06:35 /run/momentary

Verify that the systemd-tmpfiles --clean command removes from the /﻿run/momentary directory any file that is unused in the last 30 seconds, based on the systemd-tmpfiles configuration for the directory.

Create the /run/momentary/test file.

[root@servera ~]# touch /run/momentary/test

Configure your shell prompt not to return for 30 seconds.

[root@servera ~]# sleep 30

After your shell prompt returns, clean stale files from the /run/momentary directory, based on the referenced rule in the /etc/tmpfiles.d/momentary.conf configuration file.

The command removes the /run/momentary/test file, because it remains unused for 30 seconds. This behavior is based on the referenced rule in the /etc/tmpfiles.d/momentary.conf configuration file.

[root@servera ~]# systemd-tmpfiles --clean \

/etc/tmpfiles.d/momentary.conf

Verify that the /run/momentary/test file does not exist.

[root@servera ~]# ls -l /run/momentary/test

ls: cannot access '/run/momentary/test': No such file or directory

Return to the workstation machine as the student user.

[root@servera ~]# exit

logout

[student@servera ~]$ exit

logout

Connection to servera closed.

[student@workstation ~]$

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ lab finish scheduling-tempfiles

This concludes the section.

```

### Quiz: Schedule Future Tasks

```

Choose the correct answers to the following questions.

1.

Which command displays all the user jobs that you scheduled to run as deferred jobs?

A

atq

B

atrm

C

at -c

D

at --display

2.

Which command removes the deferred user job with the job number 5?

A

at -c 5

B

atrm 5

C

at 5

D

at --delete 5

3.

Which command displays all the scheduled recurring user jobs for the currently logged-in user?

A

crontab -r

B

crontab -l

C

crontab -u

D

crontab -V

4.

Which job format executes the /usr/local/bin/daily\_backup command hourly from 9 AM to 6 PM on all days from Monday through Friday?

A

00 \* \* \* Mon-Fri /usr/local/bin/daily\_backup

B

\* \*/9 \* \* Mon-Fri /usr/local/bin/daily\_backup

C

00 \*/18 \* \* \* /usr/local/bin/daily\_backup

D

00 09-18 \* \* Mon-Fri /usr/local/bin/daily\_backup

5.

Which directory contains the shell scripts to run daily?

A

/etc/cron.d

B

/etc/cron.hourly

C

/etc/cron.daily

D

/etc/cron.weekly

6.

Which configuration file defines the settings for the system jobs that run daily, weekly, and monthly?

A

/etc/crontab

B

/etc/anacrontab

C

/etc/inittab

D

/etc/sysconfig/crond

7.

Which systemd unit regularly triggers the cleanup of temporary files?

A

systemd-tmpfiles-clean.timer

B

systemd-tmpfiles-clean.service

C

dnf-makecache.timer

D

unbound-anchor.timer

```

### Summary

1. Deferred jobs or tasks are scheduled to run once in the future.

1. Recurring user jobs execute the user's tasks on a repeating schedule.

1. Recurring system jobs accomplish, on a repeating schedule, administrative tasks with system-wide impact.

1. The systemd timer units can execute both the deferred and recurring jobs.

[Next](/cH5Jk3R9Rpy-4\_RrwZQ5gA)